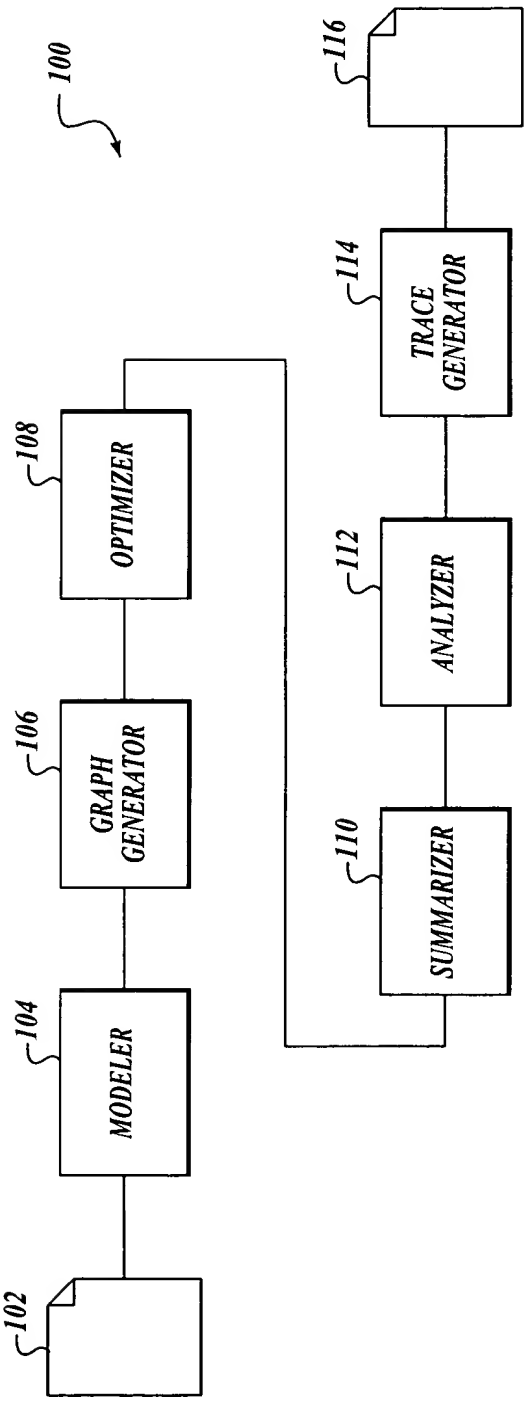

FIG. 1

200

```
int numUnits;
int level;
void getUnit() {
[1]   bool canEnter ;= F;
[2]   if (numUnits = 0) {
[3]     if (level > 10){
[4]       NewUnit();
[5]       numUnits := 1;
[6]       canEnter := T;
        }
      } else
[7]     canEnter ;= T;

[8]   if (canEnter)
[9]     if (numUnits = 0) {
[10]      assert(F);
        else
[11]      gotUnit();
      }
```

P 202

```
void getUnit() {
[1]   ...;
[2]   if (?)  {
[3]     if (?) {
[4]       ...;
[5]       ...;
[6]       ...;
        }
      } else
[7]       ...;

[8]   if (?)
[9]     if (?)
[10]      ...;
        else
[11]      ...;
      }
```

B_1 204

```
bool nU0;

void getUnit() {
[1]   ...;
[2]   if (nU0) {
[3]     if (?) {
[4]       ...;
[5]       nU0:=F;
[6]       ...;
        }
      } else
[7]       ...;

[8]   if (?) {
[9]     if (nU0)
[10]      ...;
        else
[11]      ...;
      }
```

B_2 206

```
bool nU0;

void getUnit() {
[1]   bool cE:=F;
[2]   if (nU0) {
[3]     if (?) {
[4]       ...;
[5]       nU0:=F;
[6]       cE:=T;
        }
      } else
[7]     cE:=T;

[8]   if (cE)
[9]     if (nU0)
[10]      ...;
        else
[11]      ...;
      }
```

B_3 208

FIG. 2

300

302

```
        decl g;

        main()
        begin
          decl h;
[6]       h := !g;
[7]       A(g,h);
[8]       skip;
[9]       A(g,h);
[10]      skip;
[11]      if (g) then
[12]    R; skip;
          else
[14]        skip;
          fi
        end

        A(a1,a2)
        begin
[20]      if (a1) then
[21]        A(a2,a1);
[22]        skip;
          else
[24]        g := a2;
          fi
        end
```

304

```
bebop v1.0: (c) Microsoft Corporation.
Done creating bdd variables
Done building transition relations

Label R reachable by following path:

Line 12                 State g=1 h=0
Line 11                 State g=1 h=0
Line 10                 State g=1 h=0
        Line 22         State g=1 a1=1 a2=0
                Line 24 State g=1 a1=0 a2=1
                Line 20 State g=1 a1=0 a2=1
        Line 21         State g=1 a1=1 a2=0
        Line 20         State g=1 a1=1 a2=0
Line 9                  State g=1 h=0
Line 8                  State g=1 h=0
        Line 22         State g=1 a1=1 a2=0
                Line 24 State g=1 a1=0 a2=1
                Line 20 State g=1 a1=0 a2=1
        Line 21         State g=1 a1=1 a2=0
        Line 20         State g=1 a1=1 a2=0
Line 7                  State g=1 h=0
Line 6                  State g=1
```

FIG. 3

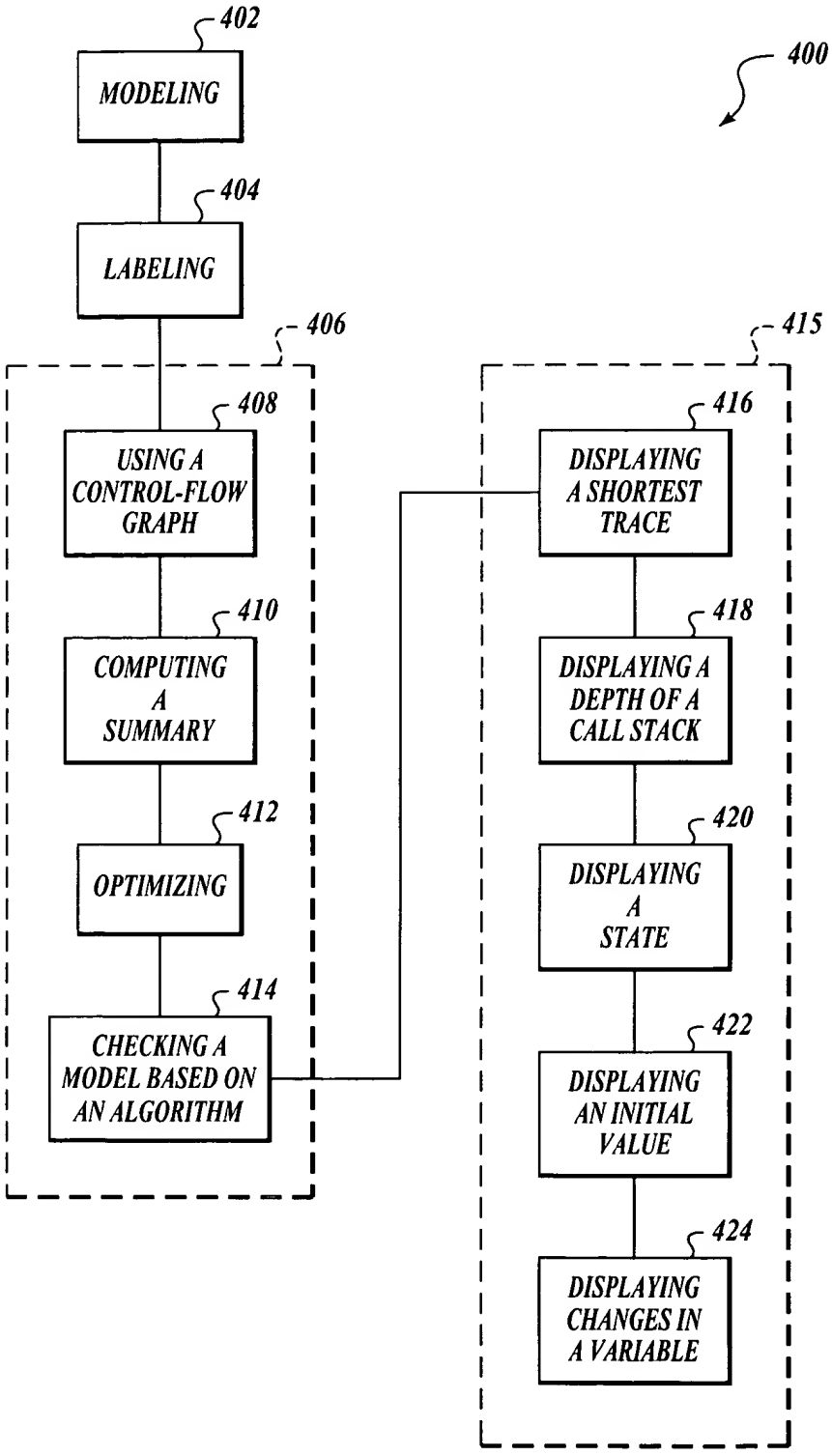
402
MODELING
400
404
LABELING
406
415
408
USING A CONTROL-FLOW GRAPH
416
DISPLAYING A SHORTEST TRACE
410
COMPUTING A SUMMARY
418
DISPLAYING A DEPTH OF A CALL STACK
412
OPTIMIZING
420
DISPLAYING A STATE
414
CHECKING A MODEL BASED ON AN ALGORITHM
422
DISPLAYING AN INITIAL VALUE
424
DISPLAYING CHANGES IN A VARIABLE

FIG. 4

500

```
[1]     if (z) {
[2]       x:=1;
        else
[3]       x:=z;
[4]     z:=y|x;
```

FIG. 5A

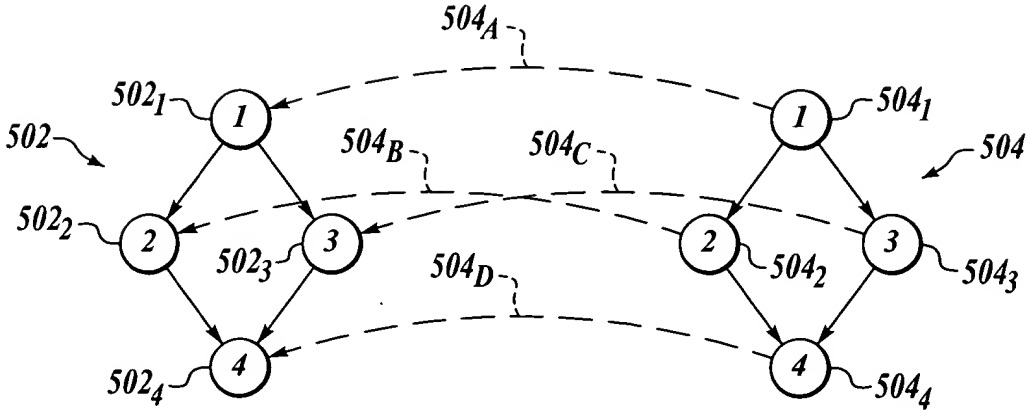

FIG. 5B

600

```
        Boolean g;
[1]     main() {
[2]       if (z) {
[3]         x:=1;
          else
[4]         x:=0;

[5]       z:=y+x;

[6]       foo (z);
[7]       skip;
[8]     }

[9]     foo (z) {
[10]      g:=1;
[11]    }
```

FIG. 6A

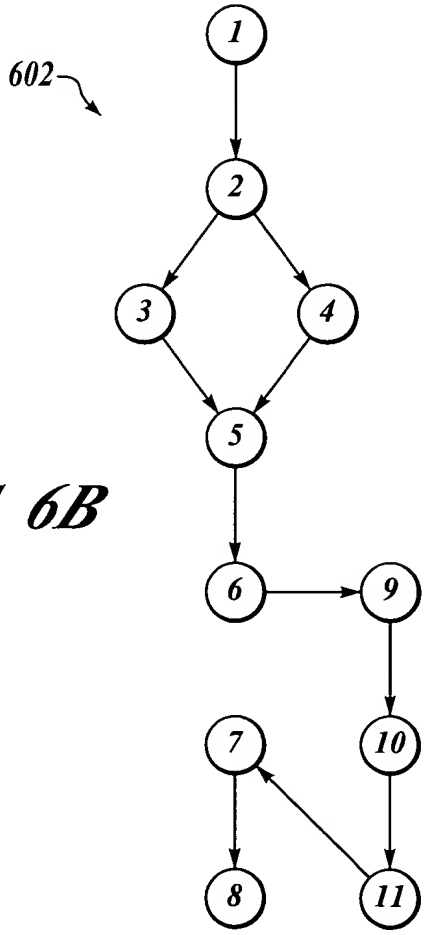

FIG. 6B

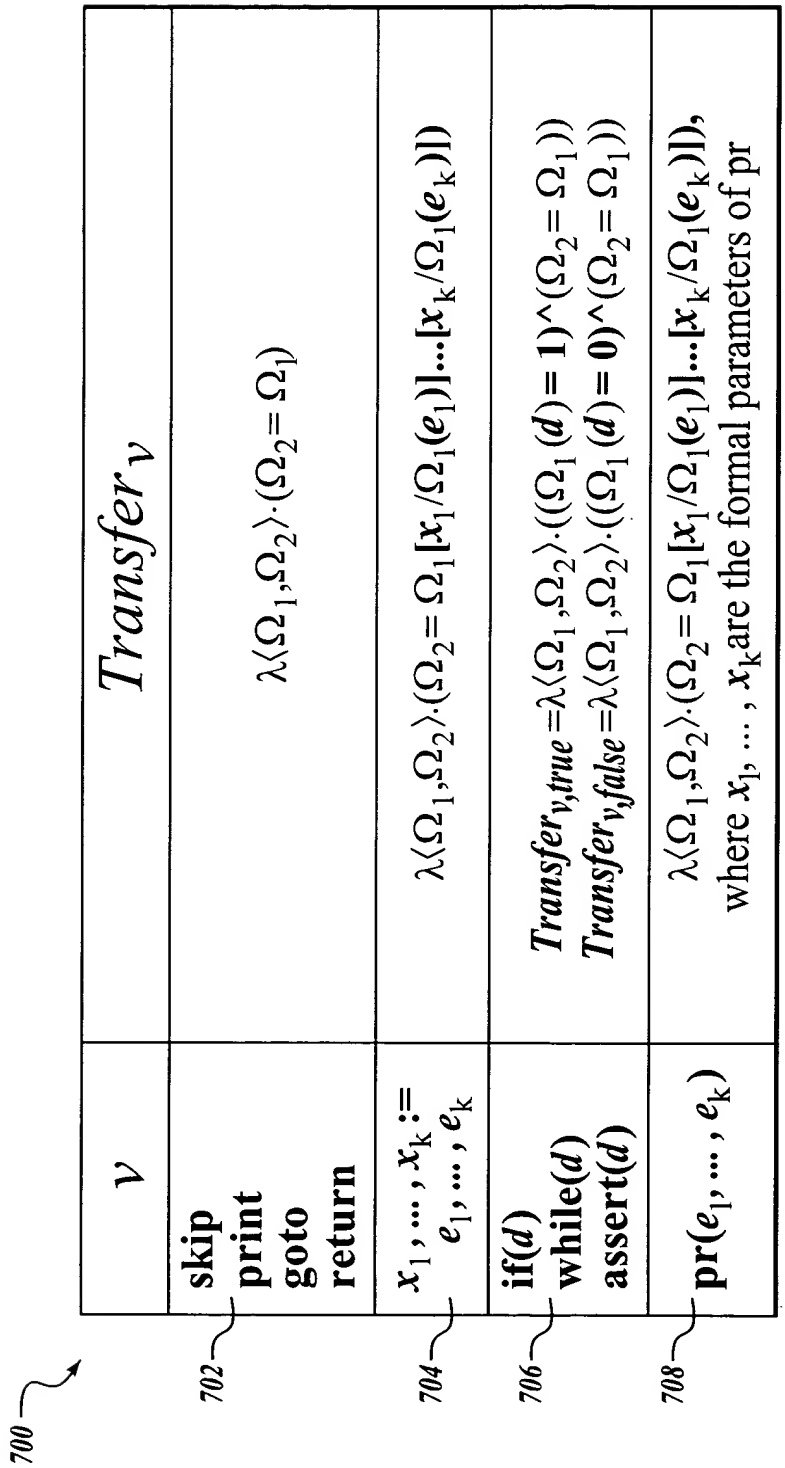
700
702
704
706
708
v
Transfer_v
skip
print
goto
return
λ⟨Ω1,Ω2⟩·(Ω2= Ω1)
x1, ... , xk := e1, ... , ek
λ⟨Ω1,Ω2⟩·(Ω2= Ω1[x1/Ω1(e1)]...[xk/Ω1(ek)])
if(d)
while(d)
assert(d)
Transfer_v,true =λ⟨Ω1,Ω2⟩·((Ω1 (d)= 1)^(Ω2= Ω1))
Transfer_v,false =λ⟨Ω1,Ω2⟩·((Ω1 (d)= 0)^(Ω2= Ω1))
pr(e1, ... , ek)
λ⟨Ω1,Ω2⟩·(Ω2= Ω1[x1/Ω1(e1)]...[xk/Ω1(ek)]),
where x1, ... , xk are the formal parameters of pr

FIG. 7

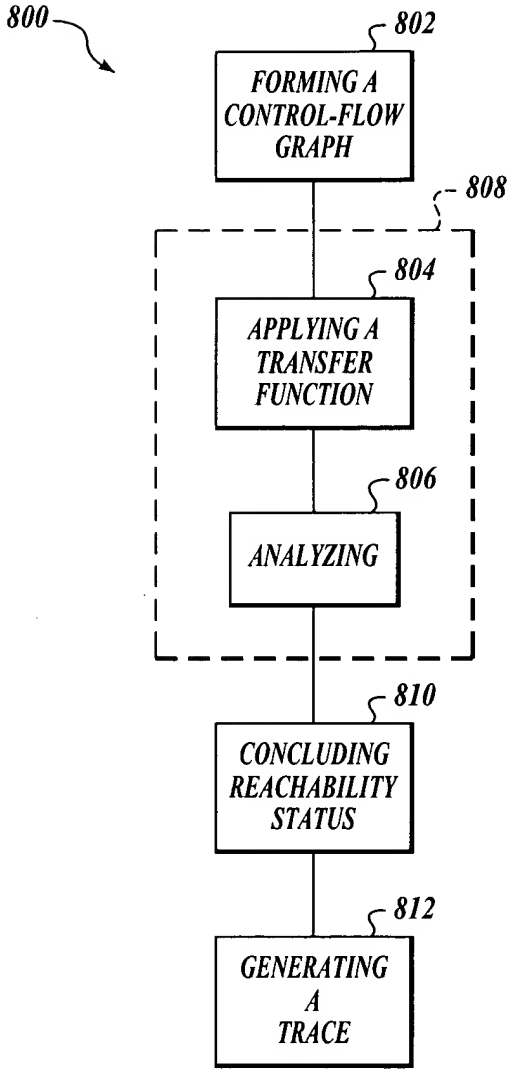
800
802
FORMING A CONTROL-FLOW GRAPH
808
804
APPLYING A TRANSFER FUNCTION
806
ANALYZING
810
CONCLUDING REACHABILITY STATUS
812
GENERATING A TRACE

FIG. 8

```
      global
902   PathEdges, SummaryEdges, WorkList                                  900

904   procedure Propagate(v,p)
      begin
906     if p ⊄ PathEdges(v) then
908       PathEdges(v) := PathEdges(v) ∪ p
910       Insert v into WorkList fi
        fi
912   end

914   procedure Reachable(G_B)
      begin
916     for all v ∈ V_B do PathEdges(v) := {}
917     for all v ∈ Call_B do SummaryEdges(v) := {}
918     PathEdges(First_B(main)) :=
            {⟨Ω, Ω⟩ | Ω is any valuation to globals and local variables of main}

920     WorkList := {First_B(main)}
922     while WorkList ≠ 0 do
924       remove vertex v from WorkList
926       switch (v)
928         case v ∈ Call_B                                                 930
              Propagate(_S Succ_B(v), SelfLoop(Join(PathEdges(v), Transfer_V)))
              Propagate(ReturnPt_B(v), Join(PathEdges(v), SummaryEdges(v)))   932
934         case v ∈ Exit_B:
              for each w ∈ Succ_B(v) do                                 936
                let
                  c ∈ Call_B such that w=ReturnPt_B(c) and              938
                  s=Lift_C(PathEdges(v), ProcOf_B(v))                   940
                in
                  if s ⊄ SummaryEdges(c) then                           944
                    SummaryEdges(c) := SummaryEdges(c) ∪ s              946
                    Propagate(w, Join(PathEdges(c), SummaryEdges(c)));  948
                ni
950         case v ∈ Cond_B:
              Propagate(Tsucc_B(v), Join(PathEdges(v), Transfer_{v,true}))   952
              Propagate(Fsucc_B(v), Join(PathEdges(v), Transfer_{v,false}))  954
956         case v ∈ V_B - Call_B - Exit_B - Cond_B:
              let p=Join(PathEdges(v), Transfer_V) in                   958
                for each w ∈ Succ_B(v) do                               960
                  Propagate(w,p)                                        962
              ni
      end
```

FIG. 9

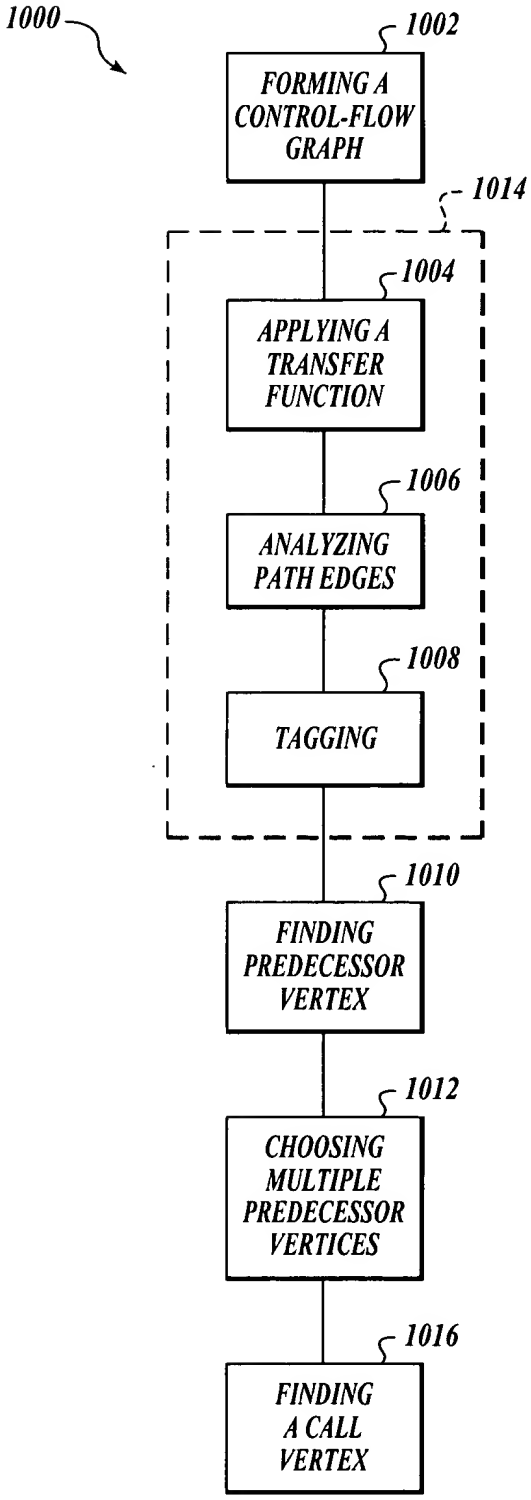
1000
1002
FORMING A CONTROL-FLOW GRAPH
1014
1004
APPLYING A TRANSFER FUNCTION
1006
ANALYZING PATH EDGES
1008
TAGGING
1010
FINDING PREDECESSOR VERTEX
1012
CHOOSING MULTIPLE PREDECESSOR VERTICES
1016
FINDING A CALL VERTEX

FIG. 10

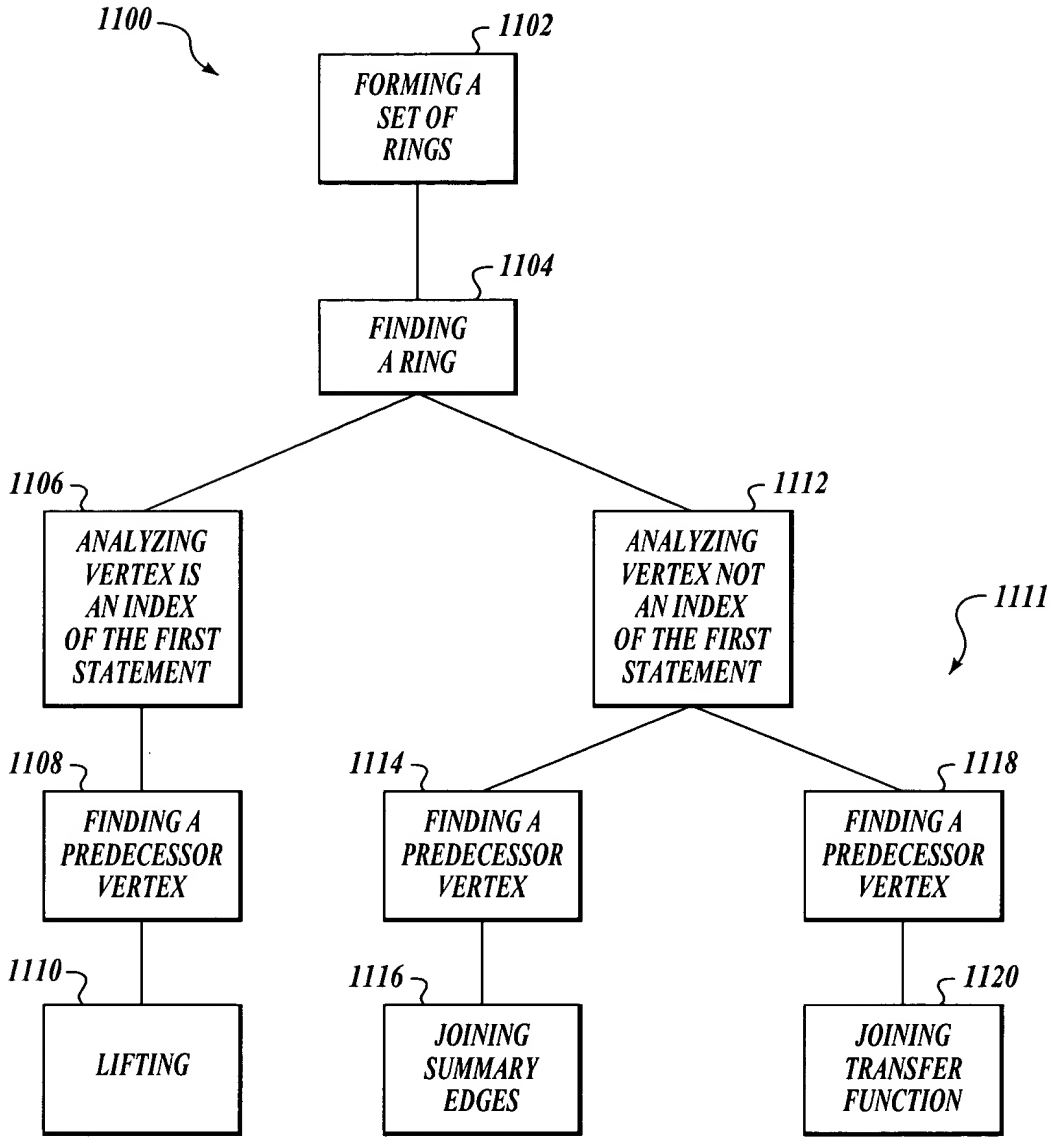
1100
1102
FORMING A SET OF RINGS
1104
FINDING A RING
1106
ANALYZING VERTEX IS AN INDEX OF THE FIRST STATEMENT
1112
ANALYZING VERTEX NOT AN INDEX OF THE FIRST STATEMENT
1111
1108
FINDING A PREDECESSOR VERTEX
1114
FINDING A PREDECESSOR VERTEX
1118
FINDING A PREDECESSOR VERTEX
1110
LIFTING
1116
JOINING SUMMARY EDGES
1120
JOINING TRANSFER FUNCTION

FIG. 11

```
global
PE' : V_G → set-of (D x D)
Worklist V_G → set-of (D x D)

procedure Propagate(v : V_G,p : (D x D))
begin
  if p ∉ PE'(v) then
     PE'(v) := PE'(v) ∪ {p}
     Worklist(v) := Worklist(v) ∪ {p}
  fi
end

procedure CMOP_SPrhs(S : set-of D)
begin
  PE'(entry) := {⟨d, d⟩ | d ∈ S}
  Worklist(entry) := PE'(entry)
  while ∃v2 s.t Worklist(v2) ≠ 0 do
    select and remove ⟨d1, d2⟩ from Worklist(v2)
    for each v2 → v3 ∈ E_G do
      for each d3 ∈ M(v2 → v3)({d2}) do
        Propagate(v3⟨d1, d3⟩)
      od
    od
  od
end
```

FIG. 12

```
global
PE' : V → set-of (set-of D x set-of D)
Worklist : V_G → set-of (set-of D x set-of D)

procedure Propagate(v : V_{G,p} : (set-of D x set-of D))
begin
  if p ∉ PE'(v) then
     PE'(v) := PE'(v) ∪ {p}
     Worklist(v) := Worklist(v) ∪ {p}
  fi
end

procedure CSMOP_{SP_rhs}(S' : set-of (set-of D))
begin
  PE'(entry) := {⟨S, S⟩ | S ∈ S'}
     Worklist(entry) := PE'(entry)
     while ∃v_2 s.t Worklist(v_2) ≠ 0 do
       select and remove ⟨S_1 , S_2⟩ from Worklist(v_2)
       for each v_2 → v_3 ∈ E_G do
         let S_3 = M( v_2 → v_3 )(S_2) in
           Propagate(v_3⟨S_1 , S_3⟩)
         ni
       od
    od
  end
```

FIG. 13